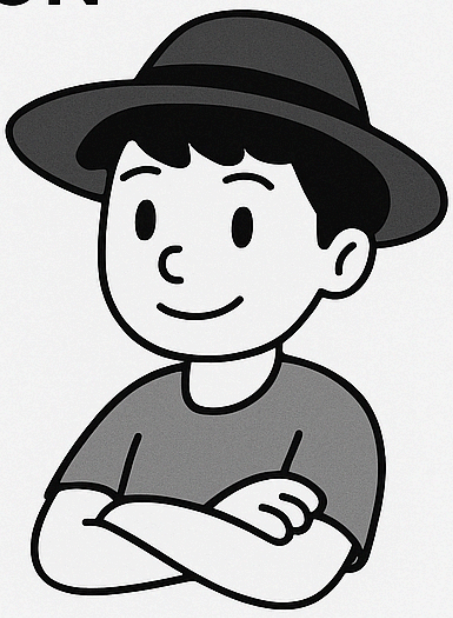
CENTRO UNIVERSITÁRIO MAURÍCIO DE NASSAU
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
1º PERÍODO

PROJETO CHICO CHAPELOTE

LÓGICA DE PROGRAMAÇÃO PYTHON



DEYVISON GUSTAVO

Centro Universitário Maurício de Nassau (UNINASSAU)

Curso Superior em Análise e Desenvolvimento de Sistemas (ADS)

Período: 1º | Aluno: Deyvison Gustavo

Data: 15 de maio de 2025

📚 PLANO DE ESTUDO COMPLETO — PYTHON DO ZERO AO AVANÇADO

✅ Módulo 1: Fundamentos do Python (Iniciante)

Objetivo: Entender a sintaxe básica e criar seus primeiros programas.

Tópicos:

1. O que é Python? Para que serve?
2. Instalação do Python e configuração do VS Code
3. Sintaxe básica: `print`, comentários, identação
4. Tipos primitivos: `int`, `float`, `str`, `bool`
5. Operadores: aritméticos, relacionais e lógicos
6. Entrada de dados com `input()` e conversão de tipos
7. Condicionais: `if`, `elif`, `else`
8. Laços de repetição: `for`, `while`
9. Listas, tuplas e dicionários (estrutura de dados)
10. Funções (`def`), parâmetros e retorno

🐍 1.1 — O que é Python?

✅ Definição

Python é uma linguagem de programação de alto nível, interpretada, de propósito geral e com sintaxe simples e legível.

Foi criada por **Guido van Rossum** e lançada em 1991. O nome **"Python"** não vem da cobra, mas do grupo de comédia britânico *Monty Python*, que Guido curtia.

🎯 Para que serve Python?

Python é **uma das linguagens mais versáteis do mundo**. Ela pode ser usada para:

Área	Aplicações comuns
Desenvolvimento Web	Sites, APIs e sistemas (com Flask ou Django)
Automação de Tarefas	Scripts para automatizar repetição de tarefas no computador
Análise de Dados	Estudo de grandes volumes de dados (com Pandas, NumPy, Matplotlib, etc.)
Inteligência Artificial	Machine Learning, Deep Learning (com TensorFlow, Scikit-Learn, PyTorch)
Jogos	Criação de jogos simples (com PyGame, por exemplo)
Aplicações Desktop	Programas com interface gráfica (com Tkinter, PyQt)
Educação	Muito usada para ensino de programação por causa da simplicidade da sintaxe

🔍 Por que Python é tão popular?

- ✅ **Sintaxe simples** (quase como escrever em inglês)
- ✅ **Gratuito e de código aberto**
- ✅ **Comunidade gigante e ativa**
- ✅ **Milhares de bibliotecas prontas**
- ✅ **Usado por gigantes da tecnologia** (Google, Netflix, NASA, Spotify...)

🧠 Curiosidade Importante

Python é **interpretado**, ou seja, **o código é lido e executado linha por linha**, em vez de ser "compilado" antes (como em Java ou C++). Isso facilita testes rápidos e torna o aprendizado mais intuitivo.

💡 Exemplo simples de código em Python:

python Copiar Editar

```python
print("Olá, Chico Chapelote! Bem-vindo ao Python!")
```

Esse código exibe uma mensagem no terminal. A função `print()` é usada para mostrar algo na tela.

📌 Resumo para fixação:

Conceito	Explicação rápida
Linguagem de programação	Conjunto de regras que permitem criar softwares
Interpretada	Executa linha por linha em tempo real
Propósito geral	Serve para diversas áreas (web, IA, automação, etc.)
Simples e legível	Ideal para quem está começando a programar

🧱 1.2 — Sintaxe básica do Python

A **sintaxe** é o conjunto de regras que definem como você deve escrever o código para que o Python entenda. Em Python, essa sintaxe é famosa por ser **limpa e intuitiva**.

🖨️ 1. `print()` — Exibir mensagens na tela

A função `print()` serve para mostrar algo no terminal:

```python
print("Aprendendo Python é divertido!")
```

Você pode imprimir:

- Textos (entre aspas)
- Números
- Variáveis
- Combinações de todos esses

Exemplos:

```python
nome = "Deyvison"
print("Olá,", nome)

idade = 23
print("Você tem", idade, "anos")
```

💬 2. Comentários

Comentários são trechos de texto no código que **não são executados**, usados para explicar o que está acontecendo. São essenciais para organização e entendimento.

▶️ Comentário de linha única:

Use o símbolo `#`

```python
# Esse programa mostra uma mensagem de boas-vindas
print("Olá, mundo!")
```

🗒️ Comentário de várias linhas (docstring):

Usa-se `"""  """` ou `'''  '''`

```python
"""
Esse programa ainda está em fase de testes.
Autor: Deyvison
"""
print("Executando...")
```

🔢 3. Identação (Indentação)

A **identação é obrigatória** em Python. Significa que **você precisa usar espaços para definir blocos de código**.

Por padrão, usamos **4 espaços** (ou uma tecla Tab, se estiver configurada assim).

```python
if True:
    print("Isso será executado")  # indentado corretamente

# ERRADO! Este bloco abaixo dará erro:
# if True:
# print("Isso vai dar erro!") ❌
```

Em outras linguagens, usamos {} para blocos.
Em Python, usamos a **indentação**.

💡 Dica:

Erros de indentação são um dos problemas mais comuns para iniciantes. Então sempre preste atenção se o seu bloco de código está corretamente recuado.

🧠 Fixando com um exemplo:

python Copiar Editar

```python
# Programa que cumprimenta o usuário
nome = "Chico Chapelote"
print("Olá,", nome)

# Testando identação com uma condição
if nome == "Chico Chapelote":
    print("Você tem um nome muito estiloso!")
```

✅ Resumo:

Elemento	Explicação
`print()`	Exibe informações na tela
`#`	Comentário de uma linha
`""" """`	Comentário de várias linhas ou docstring
Identação	Espaços usados para organizar blocos de código

🔢 1.3 — Tipos de dados e variáveis em Python

🧠 O que é uma variável?

Uma **variável** é um **espaço na memória do computador** onde você guarda um valor que pode mudar ao longo do programa.

Em Python, você cria uma variável com uma **atribuição direta**:

```python
nome = "Deyvison"
idade = 23
altura = 1.75
```

🧪 Tipos de dados mais usados:

Tipo	Exemplo	O que representa?
`str`	`"texto"`	Cadeia de caracteres (texto)
`int`	`10`, `-3`, `0`	Número inteiro
`float`	`3.14`, `-2.0`	Número decimal (ponto flutuante)
`bool`	`True` ou `False`	Valor lógico (booleano)

🧪 Exemplos práticos:

```python
nome = "Deyvison"      # str
idade = 23             # int
altura = 1.75          # float
estudando = True       # bool
```

Você pode exibir essas variáveis com `print()`:

```python
print("Nome:", nome)
print("Idade:", idade)
print("Altura:", altura)
print("Está estudando?", estudando)
```

🔍 Verificando o tipo com `type()`

```python
print(type(nome))    # <class 'str'>
print(type(idade))   # <class 'int'>
print(type(altura))  # <class 'float'>
print(type(estudando))  # <class 'bool'>
```

📝 Regras para nomes de variáveis

- ✅ Devem **começar com letra** ou **underscore (_)**
- ✅ Podem conter letras, números e underscore
- ❌ Não podem começar com número
- ❌ Não podem usar espaços ou símbolos especiais
- ❌ Não podem ter nomes iguais às **palavras reservadas** do Python (como if, while, def, etc.)

Boas práticas:

- Use **letras minúsculas**
- Para nomes compostos, use o padrão snake_case:

🧠 Fixando com um exemplo:

python Copiar Editar

```python
# Programa de informações do usuário
nome = "Deyvison"
idade = 23
altura = 1.75
estuda_programacao = True

print("Nome:", nome)
print("Idade:", idade, "anos")
print("Altura:", altura, "m")
print("Estuda programação?", estuda_programaca
```

✅ Resumo:

Conceito	Explicação breve
Variável	Armazena um valor que pode mudar
`str`	Texto
`int`	Número inteiro
`float`	Número com ponto decimal
`bool`	Verdadeiro (`True`) ou falso (`False`)
`type()`	Retorna o tipo de dado da variável

🧠 1.4 – Operadores em Python

1. Operadores Aritméticos – fazem cálculos matemáticos

Operador	Significado	Exemplo	Resultado
`+`	Adição	`3 + 2`	`5`
`-`	Subtração	`5 - 2`	`3`
`*`	Multiplicação	`4 * 3`	`12`
`/`	Divisão	`10 / 2`	`5.0`
`//`	Divisão inteira	`10 // 3`	`3`
`%`	Módulo (resto)	`10 % 3`	`1`
`**`	Exponenciação	`2 ** 3`	`8`

🧪 Exemplo:

Memórias salvas cheias ⓘ

python Copiar Editar

```python
a = 10
b = 3

print("Soma:", a + b)
print("Subtração:", a - b)
print("Multiplicação:", a * b)
print("Divisão:", a / b)
print("Divisão inteira:", a // b)
print("Resto da divisão:", a % b)
print("Potência:", a ** b)
```

2. **Operadores Relacionais (ou de comparação)** – comparam dois valores

Operador	Significado	Exemplo	Resultado
`==`	Igual a	`5 == 5`	`True`
`!=`	Diferente de	`5 != 3`	`True`
`>`	Maior que	`7 > 3`	`True`
`<`	Menor que	`2 < 5`	`True`
`>=`	Maior ou igual	`5 >= 5`	`True`
`<=`	Menor ou igual	`4 <= 3`	`False`

🧪 Exemplo:

python Copiar Editar

```python
idade = 20
print(idade == 18)  # False
print(idade >= 18)  # True
print(idade != 15)  # True
```

3. **Operadores Lógicos** – combinam condições booleanas

Operador	Significado	Exemplo	Resultado
`and`	E (todas verdadeiras)	`True and True`	`True`
`or`	OU (uma verdadeira)	`True or False`	`True`
`not`	NÃO (inverte)	`not True`	`False`

🧪 Exemplo:

python Copiar Editar

```python
maior_de_idade = True
tem_carteira = False

print(maior_de_idade and tem_carteira)   # False
print(maior_de_idade or tem_carteira)    # True
print(not tem_carteira)                  # True
```

🧠 Fixando com um exemplo completo:

python Copiar Editar

```python
idade = 20
tem_autorizacao = True

print("Pode entrar na festa?")
print(idade >= 18 and tem_autorizacao)
```

✅ Resumo

Categoria	Exemplos
Aritméticos	`+ - * / // % **`
Relacionais	`== != > < >= <=`
Lógicos	`and or not`

🧠 1.6 – Entrada de dados com `input()` e conversão de tipos

📥 O que é `input()`?

A função `input()` **permite que o usuário digite um valor** enquanto o programa está sendo executado.

```python
nome = input("Digite seu nome: ")
print("Olá,", nome)
```

Tudo que o `input()` recebe é do tipo `str` (texto), **mesmo que o usuário digite um número**.

🔁 Conversão de tipos

Para trabalhar com números, você precisa **converter a string recebida** com:

Conversão	Significado	Exemplo
`int(valor)`	Converte para inteiro	`int("10")` → `10`
`float(valor)`	Converte para decimal	`float("3.14")` → `3.14`
`str(valor)`	Converte para string	`str(25)` → `"25"`

🧪 Exemplo prático:

python Copiar Editar

```python
idade = input("Digite sua idade: ")
print("Você tem", idade, "anos")  # idade é uma string!

# Convertendo para número
idade = int(idade)
print("Daqui a 5 anos, você terá", idade + 5)
```

🎯 Exemplo completo:

python Copiar Editar

```python
nome = input("Qual é o seu nome? ")
idade = int(input("Qual é a sua idade? "))
altura = float(input("Qual é a sua altura (em metros)? "))

print("Olá,", nome)
print("Você tem", idade, "anos e", altura, "metros de altura")
```

🛑 Dica importante:

Se você tentar converter texto inválido para número, o programa dará erro:

python Copiar Editar

```python
int("abc")  # ERRO: ValueError
```

Sempre que for possível, **valide o que o usuário digitou**.

✅ Resumo

Função	Para que serve
`input()`	Recebe texto digitado pelo usuário
`int()`	Converte para número inteiro
`float()`	Converte para número com vírgula (.)
`str()`	Converte qualquer coisa em texto

🧠 1.7 – Condicionais em Python: `if`, `elif`, `else`

📌 Para que servem?

As **condicionais** permitem que o programa **tome decisões**, executando blocos de código diferentes dependendo das situações.

🧩 Sintaxe básica

```python
if condição:
    # bloco executado se a condição for verdadeira
elif outra_condição:
    # se a anterior for falsa, testa esta
else:
    # se nenhuma das anteriores for verdadeira
```

🧪 Exemplo simples:

```python
idade = int(input("Digite sua idade: "))

if idade >= 18:
    print("Você é maior de idade.")
elif idade >= 13:
    print("Você é adolescente.")
else:
    print("Você é criança.")
```

🔍 Como funciona:

- O **Python avalia a primeira condição** `if`. Se for verdadeira, ele executa o bloco e ignora os outros.
- Se a primeira for falsa, ele verifica o `elif`.
- Se **nenhuma for verdadeira**, ele executa o bloco do `else`.

⚠️ Indentação (espaços à esquerda)

A indentação (recuo) é obrigatória para que o Python saiba **quais linhas pertencem ao bloco da condição**.

python Copiar Editar

```python
if True:
    print("Isso está certo!")  # indentado corretamente
```

🔁 Combinando com operadores lógicos:

python Copiar Editar

```python
idade = 20
tem_convite = True

if idade >= 18 and tem_convite:
    print("Entrada permitida.")
else:
    print("Entrada negada.")
```

✅ Resumo

Palavra-chave	Função
`if`	Inicia uma verificação condicional
`elif`	Verifica outra condição se `if` falhar
`else`	Executa se nenhuma condição for verdadeira

🧠 1.8 – Laços de Repetição em Python: `for` e `while`

🔁 O que são laços?

Laços de repetição (ou *loops*) **executam um bloco de código várias vezes**, até que uma condição seja atingida.

🔷 `for` – Usado quando sabemos quantas vezes queremos repetir

```python
for i in range(5):
    print("Repetição número:", i)
```

🗒️ O que faz `range(n)`?

Cria uma sequência de números de `0` até `n - 1`.

```python
range(5) → [0, 1, 2, 3, 4]
```

🧪 Exemplo com `for`:

```python
nomes = ["Ana", "Bruno", "Carlos"]

for nome in nomes:
    print("Olá,", nome)
```

🔷 `while` – Usado quando **não sabemos exatamente quantas vezes vamos repetir**

```python
contador = 0

while contador < 5:
    print("Contando:", contador)
    contador += 1
```

⚠️ **Cuidado: `while` pode causar loops infinitos se a condição nunca for falsa.**

🔄 Diferença entre `for` e `while`

Situação	Melhor laço
Número fixo de repetições	`for`
Repetição até uma condição mudar	`while`

🛑 Comandos especiais dentro de laços

Comando	Função
`break`	Interrompe o laço imediatamente
`continue`	Pula a iteração atual e vai para a próxima

✅ Exemplos práticos

📍 Contar de 1 a 10:

python Copiar Editar

```python
for i in range(1, 11):
    print(i)
```

📍 Soma dos números digitados até o usuário digitar `0`:

python Copiar Editar

```python
soma = 0
numero = int(input("Digite um número (0 para parar): "))

while numero != 0:
    soma += numero
    numero = int(input("Digite outro número (0 para parar): "))

print("Soma total:", soma)
```

🧠 1.9 — Estruturas de Dados: Listas, Tuplas e Dicionários

🔹 LISTAS (`list`)

✅ O que é?

Uma **lista** é uma coleção de elementos **mutável** (pode ser alterada).

python Copiar Editar

```python
frutas = ["maçã", "banana", "uva"]
```

📌 Acessando elementos:

python Copiar Editar

```python
print(frutas[0])  # maçã
print(frutas[-1]) # uva (último elemento)
```

🔄 Modificando:

python Copiar Editar

```python
frutas[1] = "laranja"  # altera "banana" para "laranja"
```

➕ Métodos úteis:

Método	O que faz
`append(valor)`	Adiciona ao final
`insert(pos, valor)`	Insere na posição
`remove(valor)`	Remove o valor
`pop()`	Remove o último
`len(lista)`	Tamanho da lista

```python
frutas.append("melancia")
print(frutas)  # ['maçã', 'laranja', 'uva', 'melancia']
```

🔸 TUPLAS (`tuple`)

✅ O que é?

Uma **tupla** é como uma lista, **mas imutável** (não pode ser alterada).

```python
cores = ("vermelho", "verde", "azul")
```

📌 Usos:

- Quando os dados **não devem mudar**
- Mais **eficiente e seguro** que lista

```python
print(cores[1])  # verde
# cores[0] = "amarelo" → Erro!
```

🔹 DICIONÁRIOS (`dict`)

✅ O que é?

Um **dicionário** armazena dados em **pares chave-valor**.

python Copiar Editar

```python
pessoa = {
    "nome": "Deyvison",
    "idade": 22,
    "cidade": "Recife"
}
```

📌 Acessando:

python Copiar Editar

```python
print(pessoa["nome"])  # Deyvison
```

🔄 Modificando e adicionando:

python Copiar Editar

```python
pessoa["idade"] = 23
pessoa["profissão"] = "Estudante"
```

➕ Métodos úteis:

Método	O que faz
`.keys()`	Retorna todas as chaves
`.values()`	Retorna todos os valores
`.items()`	Retorna pares (chave, valor)
`del dicionario[chave]`	Remove uma chave específica

✅ Resumo

Estrutura	Mutável?	Índices?	Ideal para
Lista	Sim	Sim	Coleções mutáveis, ordem importa
Tupla	Não	Sim	Dados fixos, desempenho, segurança
Dicionário	Sim	Não	Dados nomeados (chave-valor)

🧠 1.10 — Funções (`def`), Parâmetros e Retorno

🔧 O que é uma função?

Uma **função** é um bloco de código que executa uma **tarefa específica**. Ela permite **reutilizar código**, **organizar melhor os programas** e **evitar repetições**.

✅ Definindo uma função

python Copiar Editar

```python
def saudacao():
    print("Olá! Seja bem-vindo ao Python.")
```

▶️ Chamando a função

python Copiar Editar

```python
saudacao()
```

🧩 Parâmetros

Você pode passar **valores de entrada** para uma função por meio de **parâmetros**:

python Copiar Editar

```python
def saudacao(nome):
    print(f"Olá, {nome}! Seja bem-vindo.")
```

python Copiar Editar

```python
saudacao("Deyvison")  # Olá, Deyvison!
```

Você pode usar **quantos parâmetros quiser**:

```python
def somar(a, b):
    print("Soma:", a + b)

somar(3, 5)
```

🔙 Retorno de valor (`return`)

Se quiser que a função **devolva** um resultado, use `return`:

```python
def multiplicar(a, b):
    return a * b

resultado = multiplicar(4, 5)
print("Resultado:", resultado)
```

📝 Vantagens das funções:

✅ Evitam repetição de código
✅ Facilitam manutenção
✅ Organizam melhor o programa
✅ Podem ser testadas separadamente

⚠️ Dica: `return` encerra a função!

```python
def verificar(numero):
    if numero > 0:
        return "Positivo"
    return "Negativo ou zero"
```

🧪 Exemplos práticos

python Copiar Editar

```python
def media(a, b, c):
    return (a + b + c) / 3

print("Média:", media(7, 8, 9))
```

✅ Resumo

Conceito	Exemplo
Definir	`def minha_funcao():`
Parâmetro	`def saudacao(nome):`
Retorno	`return valor`
Chamar função	`minha_funcao()`